100 Git Commands

1. **git init**
   Initializes a new Git repository in the current directory.

```
git init
```

2. **git clone**
   Creates a copy of an existing Git repository (remote or local).

```
git clone <repository-url>
```

3. **git status**
   Shows the current state of the working directory and staging area.

```
git status
```

4. **git add**
   Stages changes (files) for the next commit.

```
git add <file-name>
git add .
```

5. **git commit**
   Records changes to the repository. Use -m for a commit message.

```
git commit -m "Commit message"
```

6. **git log**
   Displays the commit history for the repository.

```
git log
```

```
git log --oneline
```

7. **git diff**
   Shows the difference between changes in the working directory and the repository.

```
git diff
```

```
git diff <commit-hash1> <commit-hash2>
```

8. **git show**
   Displays the changes introduced by a specific commit.

```
git show <commit-hash>
```

9. **git config**
   Configures user details and preferences for Git.

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

10. **git help**
    Displays help information for a Git command.

```
git help <command>
```

Branching Commands

11. **git branch**
    Lists branches, creates a new branch, or deletes a branch.

```
git branch                # List branches

git branch <branch-name>  # Create a branch

git branch -d <branch-name>  # Delete a branch
```

12. **git checkout**
    Switches to a branch or a specific commit.

```
git checkout <branch-name>

git checkout <commit-hash>
```

13. **git switch**
    Switches branches (an alternative to git checkout).

```
git switch <branch-name>
```

14. **git merge**
    Combines changes from one branch into another.

```
git merge <branch-name>
```

15. **git rebase**
    Reapplies commits from one branch onto another, rewriting history.

```
git rebase <branch-name>
```

Staging and Stashing Commands

16. **git stash**
    Temporarily saves changes that are not ready to commit.

```
git stash
```

17. **git stash pop**
    Applies the most recent stash and removes it from the stash list.

```
git stash pop
```

18. **git stash list**
    Lists all stashes saved in the repository.

```
git stash list
```

19. **git stash drop**
    Removes a specific stash entry.

```
git stash drop stash@{n}
```

20. **git stash apply**
    Applies a stash without removing it from the stash list.

```
git stash apply stash@{n}
```

Undoing Changes

21. **git reset**
    Moves the branch pointer and modifies the staging area or working directory.

```
git reset --soft HEAD~1   # Undo last commit, keep changes staged
git reset --mixed HEAD~1  # Undo last commit, unstage changes
git reset --hard HEAD~1   # Undo last commit and discard changes
```

22. **git revert**
    Creates a new commit to undo the changes of a previous commit.

```
git revert <commit-hash>
```

23. **git clean**
    Removes untracked files and directories.

```
git clean -f          # Remove untracked files

git clean -fd         # Remove untracked files and directories
```

Viewing History

24. **git reflog**
    Displays a log of all reference changes (branch and HEAD movements).

```
git reflog
```

25. **git blame**
    Shows who made changes to each line of a file.

```
git blame <file-name>
```

26. **git shortlog**
    Summarizes commit history by author.

```
git shortlog
```

27. **git log --graph**
    Displays a visual representation of the commit history.

```
git log --graph
```

28. **git show <commit-hash>**
    Displays detailed information about a specific commit.

Collaboration Commands

29. **git remote**
    Manages connections to remote repositories.

```
git remote add <name> <url>
git remote remove <name>
```

30. **git fetch**
    Downloads changes from a remote repository without applying them.

```
git fetch
```

31. **git pull**
    Fetches changes from a remote repository and merges them into the current branch.

```
git pull
```

32. **git push**
    Uploads local commits to a remote repository.

```
git push origin <branch-name>
```

33. **git clone**
    Clones a remote repository to your local machine.

Cherry-Picking and Rebasing

34. **git cherry-pick**
    Applies specific commits from one branch to another.

```
git cherry-pick <commit-hash>
```

35. **git rebase --interactive**
    Squash, edit, or reorder commits during a rebase.

```
git rebase -i HEAD~3
```

Advanced Commands

36. **git bisect**
    Finds the commit that introduced a bug using binary search.

```
git bisect start
git bisect bad
git bisect good
```

37. **git tag**
    Creates, lists, or deletes tags for specific commits.

```
git tag <tag-name>
git tag -a <tag-name> -m "Message"
```

38. **git archive**
    Creates an archive of the repository files.

```
git archive --format=zip HEAD > repo.zip
```

39. **git submodule**
    Manages submodules (nested repositories).

```
git submodule add <repo-url>
```

```
git submodule update
```

40. **git worktree**
    Adds multiple working directories for the same repository.

```
git worktree add <path> <branch>
```

Aliases and Shortcuts

41. **git alias**
    Creates shortcuts for frequently used commands.

```
git config --global alias.co checkout
```

```
git config --global alias.br branch
```

Debugging Commands

42. **git diff**
    Compares changes between different commits, branches, or working directories.

```
git diff <commit-hash1> <commit-hash2>
git diff main feature
```

43. **git log --stat**
    Shows commit history with a summary of changes for each commit.

```
git log --stat
```

44. **git grep**
    Searches for a string or pattern in your repository.

```
git grep "search-string"
```

45. **git bisect run**
    Automates git bisect using a script to test each commit.

```
git bisect run <script>
```

46. **git fsck**
    Verifies the integrity of the Git repository.

```
git fsck
```

47. **git log -S**
    Searches for commits where a string was added or removed.

```
git log -S "search-term"
```

48. **git show-branch**
    Displays the branch history in a compact form.

```
git show-branch
```

49. **git rev-parse**
    Converts branch names or tags into commit hashes.

```
git rev-parse HEAD
```

50. **git whatchanged**
    Shows the file-level changes for each commit.

```
git whatchanged
```

Patch Management

51. **git format-patch**
    Generates patch files from commits.

```
git format-patch HEAD~3
```

52. **git apply**
    Applies a patch file to the working directory.

```
git apply <patch-file>
```

53. **git am**
    Applies patches and creates commits from them.

```
git am <patch-file>
```

54. **git diff --cached**
    Shows the differences between the staging area and the last commit.

```
git diff --cached
```

55. **git reset HEAD**
    Unstages a file, moving it back to the working directory.

```
git reset HEAD <file>
```

Collaborative Workflow Commands

56. **git push --force**
    Force pushes changes to a remote branch (used cautiously).

```
git push --force
```

57. **git fetch --prune**
    Cleans up deleted remote branches locally.

```
git fetch --prune
```

58. **git pull --rebase**
    Rebases instead of merging during a pull operation.

```
git pull --rebase
```

59. **git remote prune**
    Removes references to deleted remote branches.

```
git remote prune origin
```

60. **git cherry**
    Lists commits in the current branch that are not in the target branch.

```
git cherry main feature
```

Rewriting History

61. **git rebase --onto**
    Rebases a range of commits onto a different branch.

```
git rebase --onto <new-base> <upstream> <branch>
```

62. **git filter-branch**
    Rewrites commit history for advanced filtering.

```
git filter-branch --env-filter '...' HEAD
```

63. **git replace**
    Replaces a commit with another.

```
git replace <commit-hash1> <commit-hash2>
```

64. **git reset --merge**
    Resets the working directory and index, preserving uncommitted changes.

```
git reset --merge
```

65. **git commit --amend**
    Modifies the most recent commit message or adds changes to it.

```
git commit --amend
```

Advanced Commands

66. **git reflog expire**
    Clears old or unnecessary reflog entries.

```
git reflog expire --all --expire=now
```

67. **git gc**
    Cleans up unnecessary files and optimizes the repository.

```
git gc
```

68. **git worktree prune**
    Cleans up old worktree references.

```
git worktree prune
```

69. **git prune**
    Cleans up unreachable objects.

```
git prune
```

70. **git ls-tree**
    Displays the content of a tree object in Git.

```
git ls-tree HEAD
```

Security and Authentication

71. **git credential**
    Manages Git credentials.

```
git credential approve
```

72. **git config --list**
    Lists all configuration settings.

```
git config --list
```

73. **git verify-commit**
    Verifies signed commits.

```
git verify-commit <commit-hash>
```

Git Hooks

74. **git hook**
    Hooks are custom scripts triggered by Git actions.
    - Examples: pre-commit, post-merge.
    - Place scripts in the .git/hooks/ directory.
75. **git commit-msg**
    A hook to validate or modify commit messages.

Logging and Auditing

76. **git log --since**
    Shows commits since a specific date.

```
git log --since="2 weeks ago"
```

77. **git log --author**
    Filters commits by author.

```
git log --author="John Doe"
```

78. **git log --grep**
    Filters commits by a commit message pattern.

```
git log --grep="bug fix"
```

79. **git log --patch**
    Displays the patch introduced by each commit.

```
git log --patch
```

80. **git log --decorate**
    Shows commits with references (branches, tags).

```
git log --decorate
```

Aliases and Shortcuts

81. **git alias**
    Creates aliases for commands to simplify workflows.

```
git config --global alias.co checkout
```

82. **git config --edit**
    Opens the Git configuration file for editing.

```
git config --edit
```

83. **git custom**
    You can create custom commands by defining shell scripts and placing them in your PATH.

Collaboration Commands

84. **git remote rename**
    Renames a remote repository.

```
git remote rename <old-name> <new-name>
```

85. **git push --set-upstream**
    Sets the upstream branch for the current branch and pushes it.

```
git push --set-upstream origin <branch-name>
```

86. **git pull origin <branch>**
    Fetches and merges changes from the specified branch on the remote.

```
git pull origin main
```

87. **git push origin --delete <branch>**
    Deletes a remote branch.

```
git push origin --delete feature-branch
```

88. **git remote show**
    Displays detailed information about a remote repository.

```
git remote show origin
```

89. **git push --tags**
    Pushes all local tags to the remote repository.

```
git push --tags
```

90. **git fetch --all**
    Fetches updates from all remotes.

```
git fetch --all
```

91. **git log origin/main..HEAD**
    Shows commits that are on the current branch but not in the remote branch.

```
git log origin/main..HEAD
```

Git Attributes and Ignore Files

92. **git ls-files**
    Lists tracked files.

```
git ls-files
```

93. **.gitignore**
    Specifies files or directories for Git to ignore.
    - Example:

```
# Ignore log files
*.log
```

94. **git check-ignore**
    Checks whether a file is ignored based on .gitignore.

```
git check-ignore <file>
```

95. **git add -f**
    Adds an ignored file forcefully.

```
git add -f <file>
```

96. **git attributes**
    Customizes handling of specific files in the repository via .gitattributes.
    - Example:

```
*.jpg binary
*.txt diff
```

Automation and Workflows

97. **git rebase -i HEAD~n**
    Interactively rebases the last n commits for squashing, editing, or reordering.

```
git rebase -i HEAD~3
```

98. **git cherry-pick --no-commit**
    Cherry-picks a commit without creating a commit immediately.

```
git cherry-pick --no-commit <commit-hash>
```

99. **git stash save "message"**
    Saves changes to the stash with a description.

```
git stash save "Work-in-progress: feature X"
```

100. **git sparse-checkout**
    Checks out a subset of the repository.
    ```
    git sparse-checkout init git sparse-checkout set <path>
    ```

Complete Categories Overview

Category	**Commands**
Basic Setup	git init, git config, git clone
Branching	git branch, git checkout, git switch, git merge, git rebase
Staging & Committing	git add, git commit, git status, git diff
Undo Changes	git revert, git reset, git stash
History	git log, git reflog, git blame, git shortlog
Collaboration	git fetch, git pull, git push, git remote
Advanced Features	git bisect, git cherry-pick, git rebase --onto, git filter-branch
Debugging	git grep, git show, git fsck

Category	Commands
Patch Management	git format-patch, git apply, git am
Tags and Releases	git tag, git archive
Cleanup	git gc, git prune, git clean